Data Analysis and Machine Learning using Python

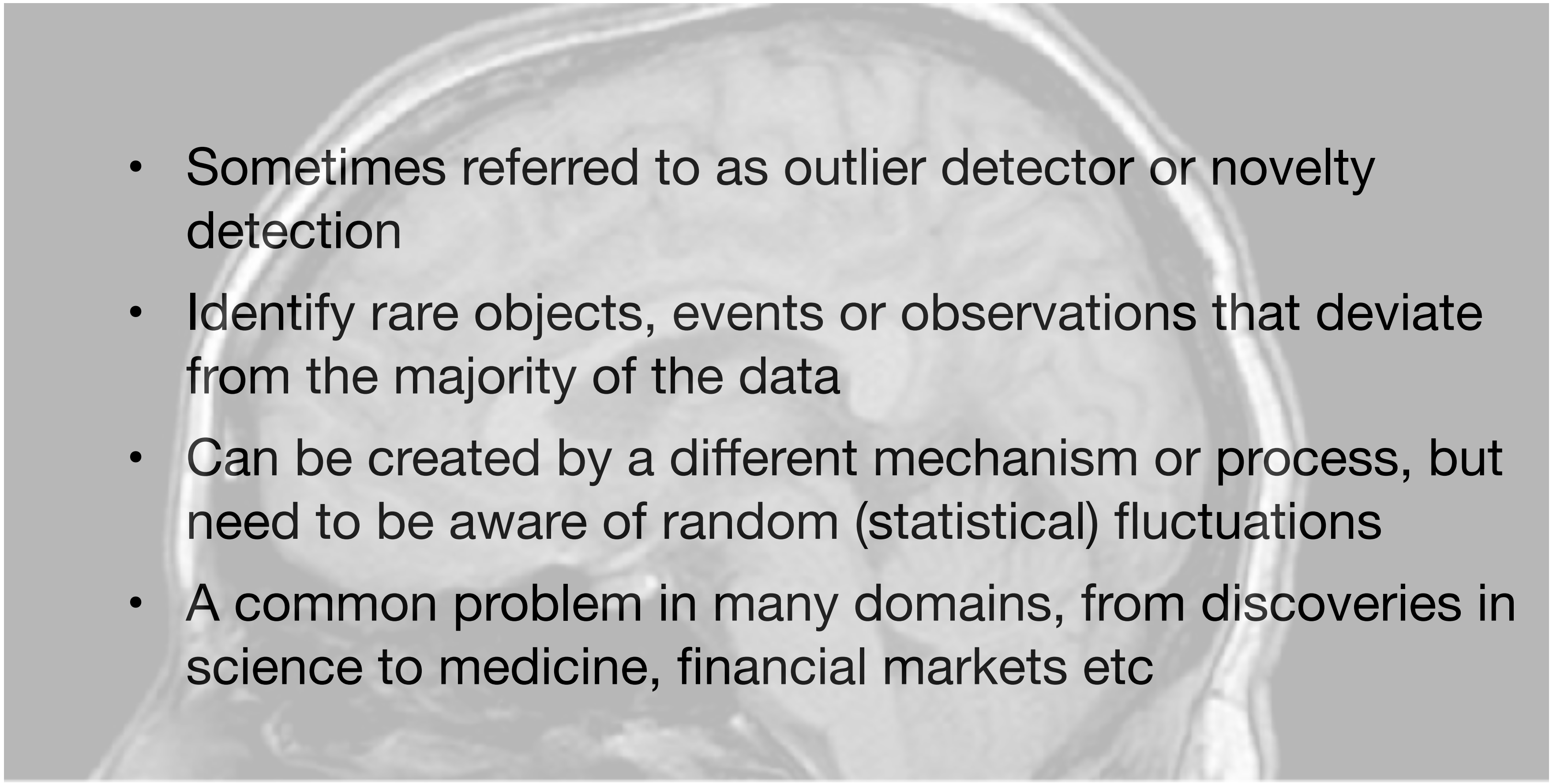
Lecture 9: Autoencoders, Deep Neural Networks, Pytorch, Anomaly detection, Reinforcement learning

May 11 2024

Today

- Homework 7
- Anomaly detection - idea
- Autoencoders
- PyTorch - absolute basics
- Example of an autoencoder in PyTorch
- Reinforcement learning - ideas
- Reinforcement learning - example

Anomaly detection



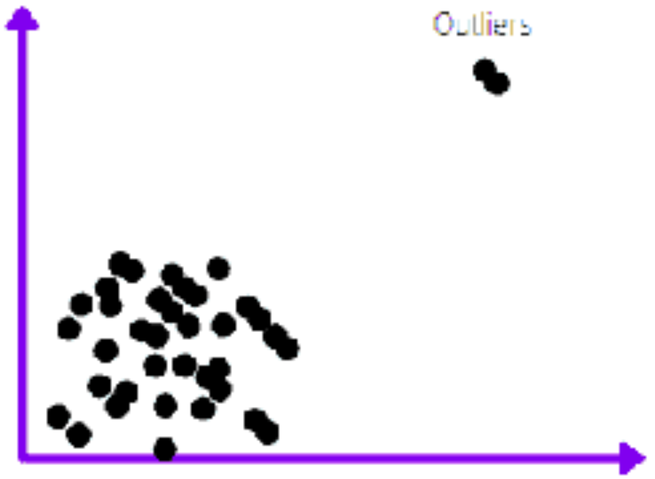
- Sometimes referred to as outlier detector or novelty detection
- Identify rare objects, events or observations that deviate from the majority of the data
- Can be created by a different mechanism or process, but need to be aware of random (statistical) fluctuations
- A common problem in many domains, from discoveries in science to medicine, financial markets etc

Anomaly detection

- Three basic categories:
  - Supervised: just like for other classification problems, have labeled training set of normal and anomalous objects and train classifier. Then apply classification on unknown data
  - Semi-supervised: some small fraction of labeled data, often only “normal” data
  - Unsupervised: Only unlabeled data. This is the most common application of automated anomaly detection

Anomaly detection

- Many methods have been proposed.
- Obviously, clustering can play a role here:

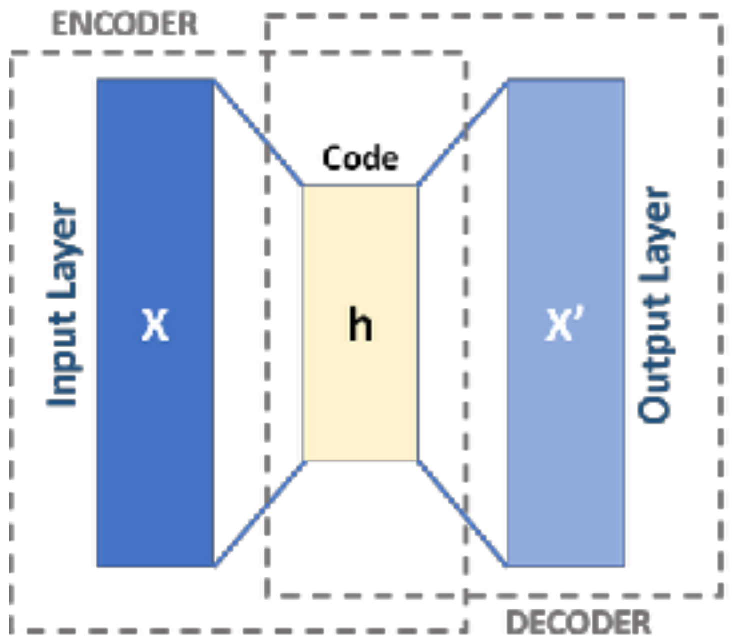


- As always, this will be much more difficult in a high-dimensional space

Autoencoders

- Autoencoders are a class of neural networks that can be employed for anomaly detection
- They combine many of the concepts we have discussed:
  - deep neural networks
  - dimensionality reduction
  - unsupervised learning
  - anomaly detection

What is an autoencoder?

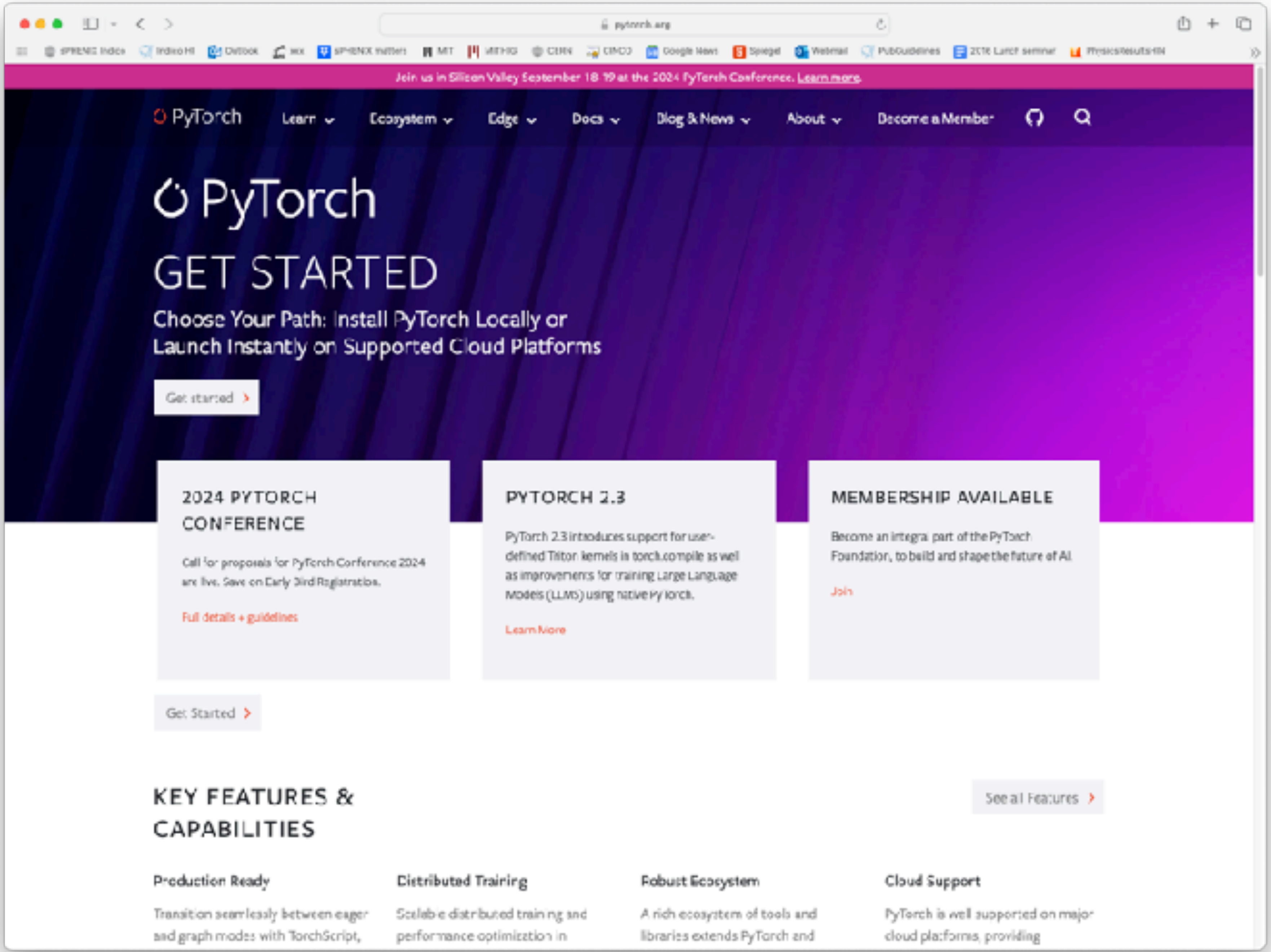


- Autoencoder transfers input X to output X'
- Two neural networks: Encoder and Decoder
- Key is the intermediate Code layer **h** which generally has **lower dimension** than the input layer
- This forces the Encoder to learn how to **reduce dimensionality** of the input data
- Train autoencoder to make output X' as close to X as possible

Applications of auto encoders

- Image/feature detection: Represent image data or image features in low dimensional space for image/ feature recognition
- Anomaly detection: Detect subsets of data that are dissimilar to the bulk of the data
- Generative models: Generate new data that is similar to the input data
- Information retrieval: Train auto encoder to produce binary code that can be used a hash table to retrieve full entries

  – We will example of auto encoder for image recognition in Pytorch



Join us in Silicon Valley September 18-19 at the 2024 PyTorch Conference. Learn more.

PyTorch

GET STARTED

Choose Your Path: Install PyTorch Locally or Launch Instantly on Supported Cloud Platforms

2024 PYTORCH CONFERENCE

Call for proposals for PyTorch Conference 2024 are live. Save on Early Bird Registration.

Full details + guidelines

PYTORCH 2.3

PyTorch 2.3 introduces support for user-defined Triton kernels in torch.compile as well as improvements for training Large Language Models (LLMs) using native PyTorch.

Learn More

MEMBERSHIP AVAILABLE

Become an integral part of the PyTorch Foundation, to build and shape the future of AI.

Join

KEY FEATURES & CAPABILITIES

Production Ready

Transition seamlessly between eager and graph modes with TorchScript,

Distributed Training

Scalable distributed training and performance optimization in

Robust Ecosystem

A rich ecosystem of tools and libraries extends PyTorch and

Cloud Support

PyTorch is well supported on major cloud platforms, providing

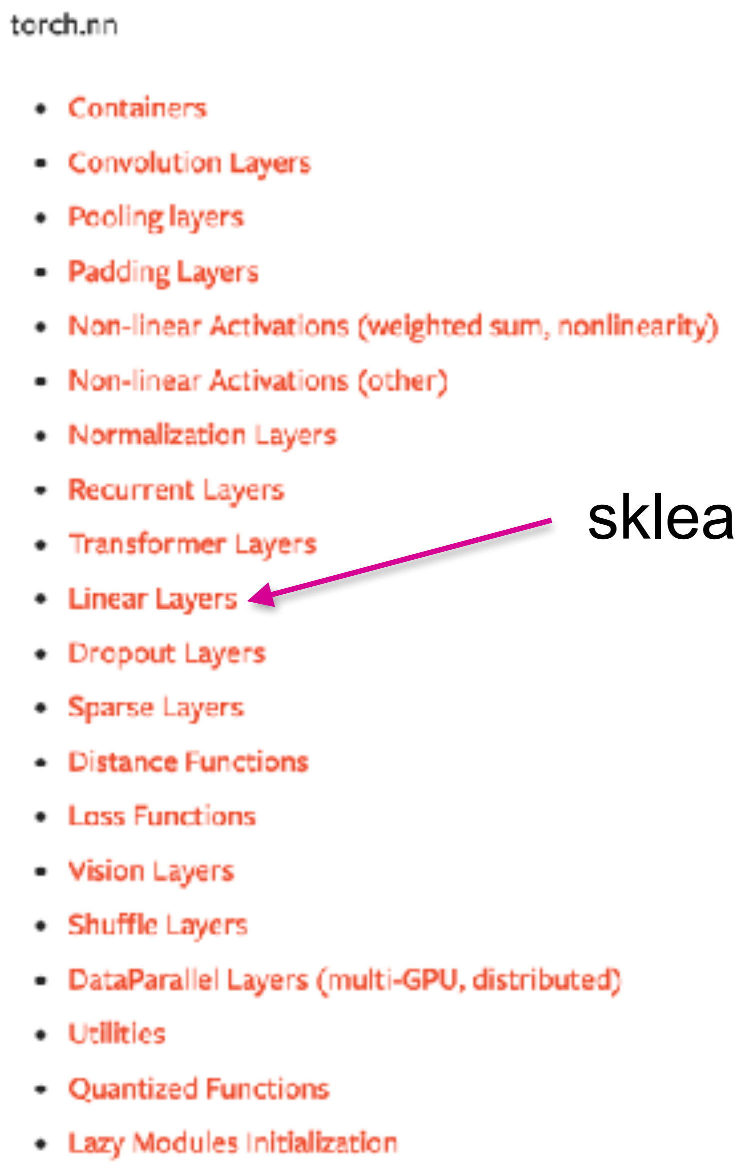
What is PyTorch?

- PyTorch is an open source, fully featured framework for building **deep learning** models
- It is based on the Python programming language and the Torch library.
- The Torch package contains data structures for multi-dimensional tensors and defines mathematical operations on these tensors.
  - In this context, just think of a tensor as a multi-dimensional array
  - PyTorch provides access to GPU acceleration of tensor operations, e.g., through integration with CUDA. GPUs provide fast, parallel execution of tensor manipulation

Deep learning, deep neural networks

- Why do we need PyTorch when we have sklearn?
- What is the difference between a deep neural network (DNN) and an MLP?
- In principle, an MLP is an example of a DNN
- But typically MLPs will have only a few hidden layers, while DNNs in general can have 1000’s
  - MLP layers are linear layers (all nodes connected), not the case in general DNNs
  - MLPs typically initialized with random weights. This does not work in case of many hidden layers. E.g., weight updates through recursive backpropagation involves multiplication of weights. If weights < 1, numerical values vanish, if weights > 1, numerical values explode.
- DNN: Need more diverse options for layers and more sophisticated initialization of weights; GPU support

DNN layer types in PyTorch

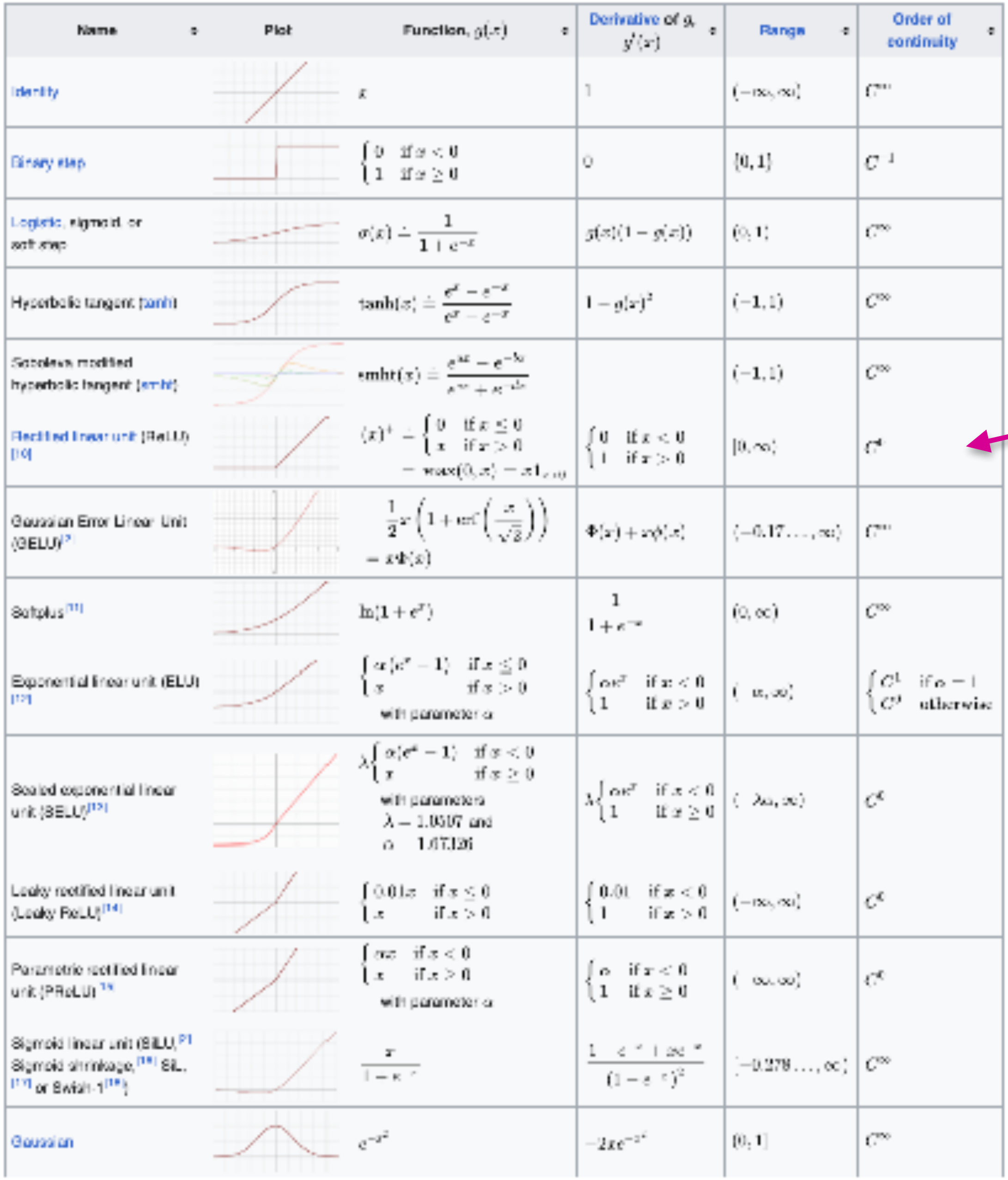


torch.nn

- Containers
- Convolution Layers
- Pooling layers
- Padding Layers
- Non-linear Activations (weighted sum, nonlinearity)
- Non-linear Activations (other)
- Normalization Layers
- Recurrent Layers
- Transformer Layers
- Linear Layers
- Dropout Layers
- Sparse Layers
- Distance Functions
- Loss Functions
- Vision Layers
- Shuffle Layers
- DataParallel Layers (multi-GPU, distributed)
- Utilities
- Quantized Functions
- Lazy Modules Initialization

sklearn MLP

Activation functions (not all shown)



Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty,\infty)$	C^∞
Binary step		$\begin{cases}0 & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$	0	$\{0,1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$	$(0,1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x-e^{-x}}{e^x+e^{-x}}$	$1-g(x)^2$	$(-1,1)$	C^∞
Soboleva modified hyperbolic tangent (smht)		$\operatorname{smht}(x) \doteq \frac{e^{ax}-e^{-bx}}{e^{cx}+e^{-dx}}$		$(-1,1)$	C^∞
Rectified linear unit (ReLU)[10]		$(x)^+ \doteq \begin{cases}0 & \text{if } x\leq 0\\ x & \text{if } x>0\end{cases}$ $=\max(0,x)=x\mathbf{1}_{x>0}$	$\begin{cases}0 & \text{if } x<0\\ 1 & \text{if } x>0\end{cases}$	$[0,\infty)$	C^0
Gaussian Error Linear Unit (GELU)[7]		$\frac{1}{2}x\left(1+\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $=x\Phi(x)$	$\Phi(x)+x\phi(x)$	$(-0.17\ldots,\infty)$	C^∞
Softplus[11]		$\ln(1+e^x)$	$\frac{1}{1+e^{-x}}$	$(0,\infty)$	C^∞
Exponential linear unit (ELU)[12]		$\begin{cases}\alpha(e^x-1) & \text{if } x\leq 0\\ x & \text{if } x>0\end{cases}$ with parameter α	$\begin{cases}\alpha e^x & \text{if } x<0\\ 1 & \text{if } x>0\end{cases}$	$(-\alpha,\infty)$	$\begin{cases}C^1 & \text{if } \alpha=1\\ C^0 & \text{otherwise}\end{cases}$
Scaled exponential linear unit (SELU)[13]		$\lambda\begin{cases}\alpha(e^x-1) & \text{if } x<0\\ x & \text{if } x\geq 0\end{cases}$ with parameters $\lambda=1.0507$ and $\alpha=1.67326$	$\lambda\begin{cases}\alpha e^x & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$	$(-\lambda\alpha,\infty)$	C^0
Leaky rectified linear unit (Leaky ReLU)[14]		$\begin{cases}0.01x & \text{if } x\leq 0\\ x & \text{if } x>0\end{cases}$	$\begin{cases}0.01 & \text{if } x<0\\ 1 & \text{if } x>0\end{cases}$	$(-\infty,\infty)$	C^0
Parametric rectified linear unit (PReLU)[15]		$\begin{cases}\alpha x & \text{if } x<0\\ x & \text{if } x\geq 0\end{cases}$ with parameter α	$\begin{cases}\alpha & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$	$(-\infty,\infty)$	C^0
Sigmoid linear unit (SiLU,[7] Sigmoid shrinkage,[16] SiL,[17] or Swish-1[18])		$\frac{x}{1+e^{-x}}$	$\frac{1+e^{-x}+xe^{-x}}{(1+e^{-x})^2}$	$[-0.278\ldots,\infty)$	C^∞
Gaussian		e^{-x^2}	$-2xe^{-x^2}$	$(0,1]$	C^∞

Example will use ReLU

PyTorch basics

- Most fundamental: PyTorch **tensors**
  - n-dimensional array, conceptually like NumPy array
  - Many function operating on operations
  - GPU support for tensor operations

- **Autograd**: Automatic differentiation to compute backward passes in neural networks
  - **forward** function of autograd operators calculates output tensors from input tensors
  - **backward** function computes gradient of output tensors with respect to some scalar value, and computes the gradient of the input tensors with respect to that same scalar value

PyTorch basics

- **Modules**: torch.nn provides the torch.nn.module base class for building neural networks
  - access to different types of NN layers
  - access to different types of loss functions
  - many utilities
- **Device:** torch.device object decides where the tensor lives - e.g., on the CPU or on GPU via the CUDA interface. Can run the same code on CPU or GPU (if one exists)

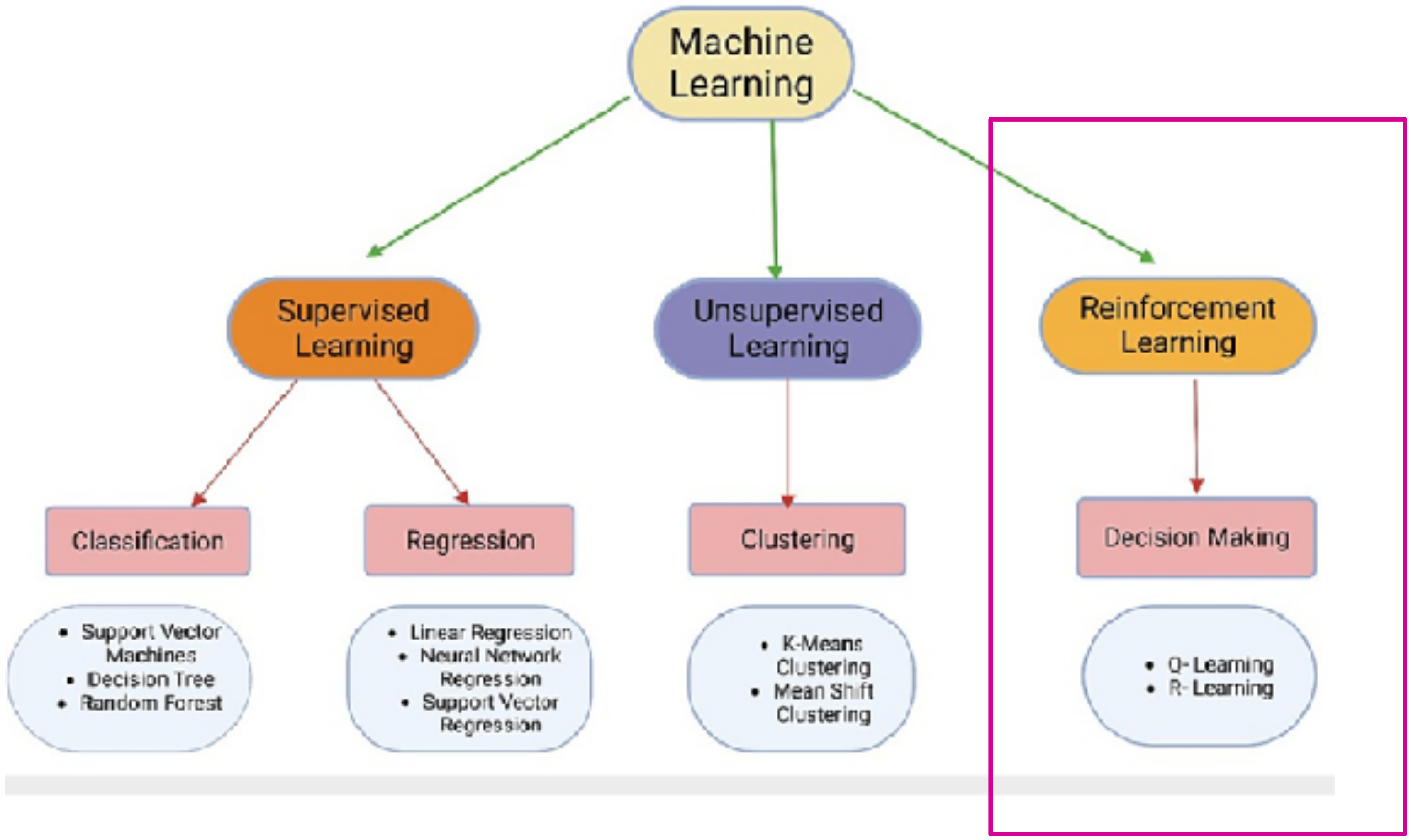
PyTorch example in detail

- Use the same set of handwritten digits that we used with an MLP classifier before
- This time we will use an autoencoder:
  - Load the data file
  - Start with the input image with 28x28 pixels (784 features)
  - use encoder model with two hidden layers to encode 784 feature input data in a 2-feature hidden layer
  - use decode model with two hidden layers to translate 2-feature hidden layer into 28x28 pixel output layer
  - Can look at the representation of the 10 different digits in the 2D latent layer
  - Can use points in the 2D space to generate new digit images

Anomaly detection with *Autoencoders*

- How can we use this for anomaly detection?
- If the anomalous event is different from the normal events that the encoder knows how to compress
- → compression (dimensionality reconstruction) will not work as well
- → decoded features X' will be more dissimilar from X than for normal events

Categories of ML

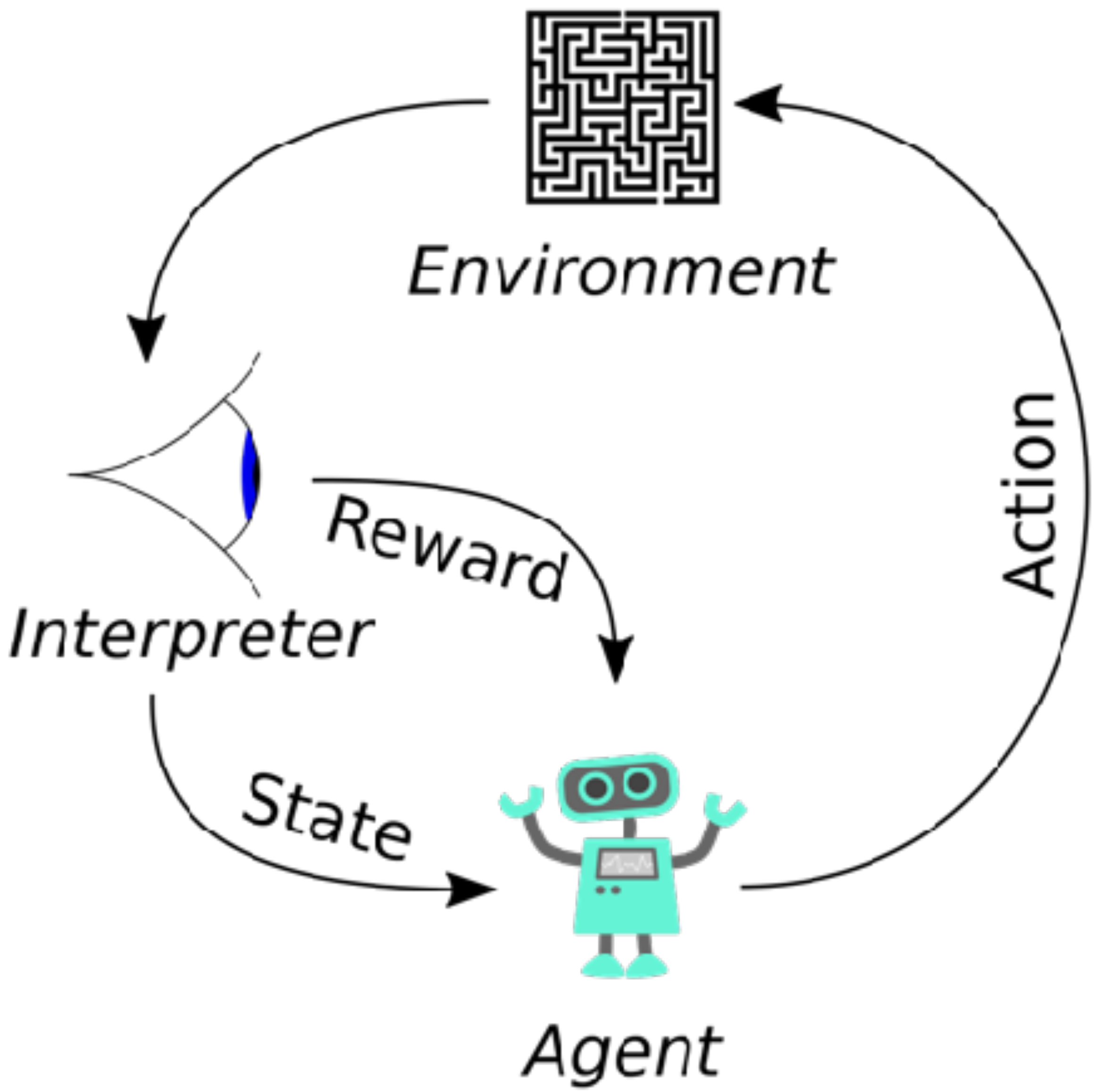


from Shruti Singh

Reinforcement learning

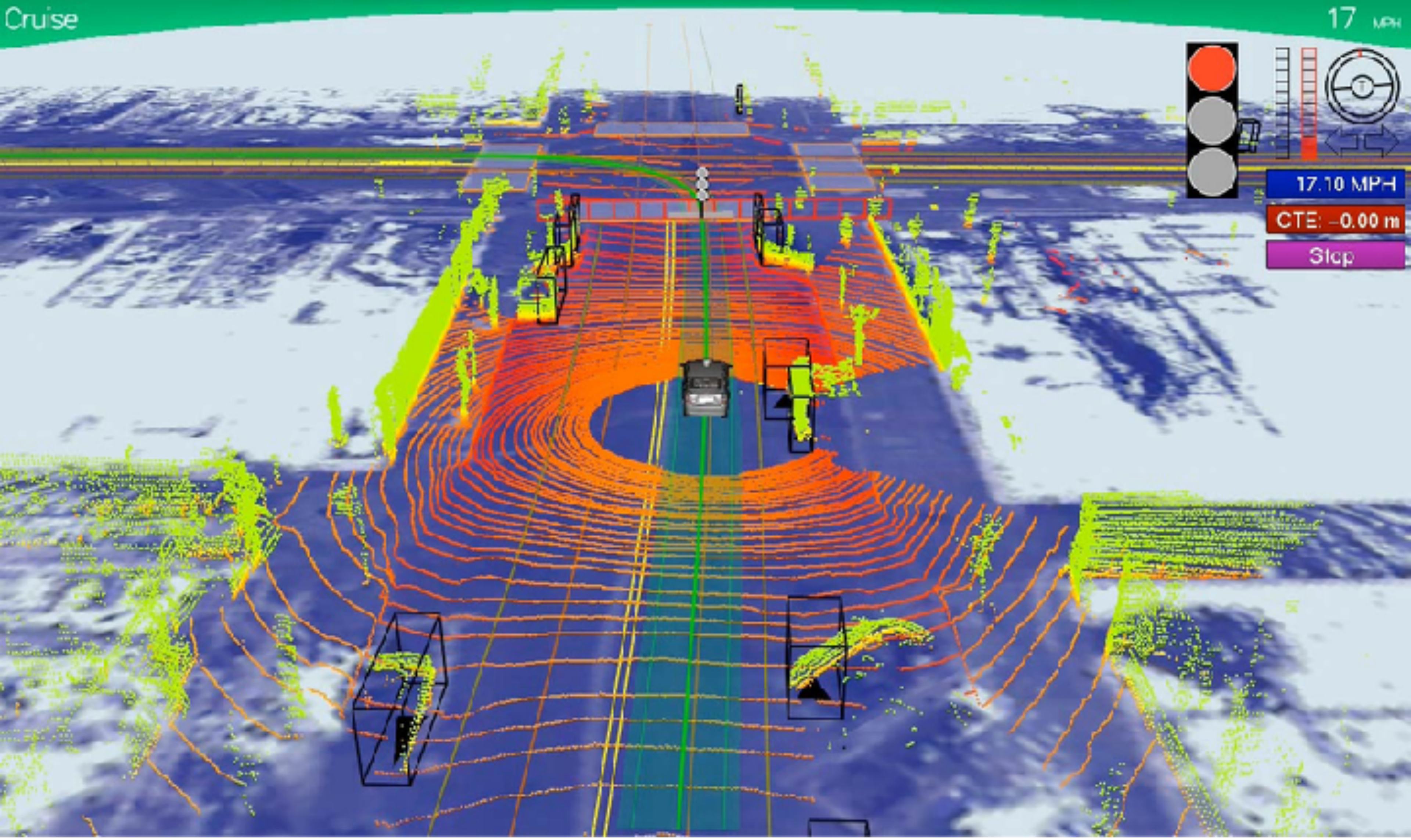
- We talked about supervised and unsupervised learning
  - note that for the autoencoder training in the PyTorch example we did not use the labels in training → unsupervised
- Reinforcement learning is another class of ML approaches
  - does not use labeled input/output data (no “supervision”)
  - tries to maximize a **long-term reward** by optimizing **policy** guiding actions of an **agent** in acting in an **environment**
  - The data the agent receives depend on agents **actions**
  - Time-ordered sequence

Elements of a RL model



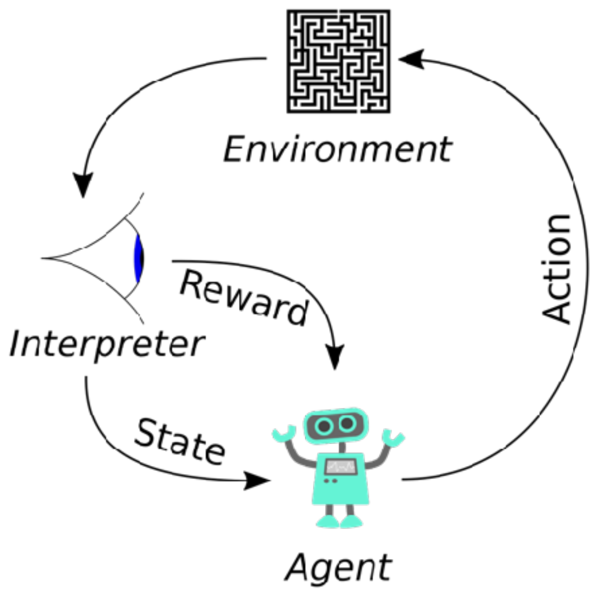
Reward R_t

- Goal is to maximize **total future reward**
- Reward Rt is a scalar feedback signal the agent receives, depending on how well agent is doing at step t
- Agent needs to take **sequence of decisions** to maximize total future reward
  - e.g., maximize total value of stock portfolio at future time
- Sometimes need to sacrifice immediate reward to maximize total future reward



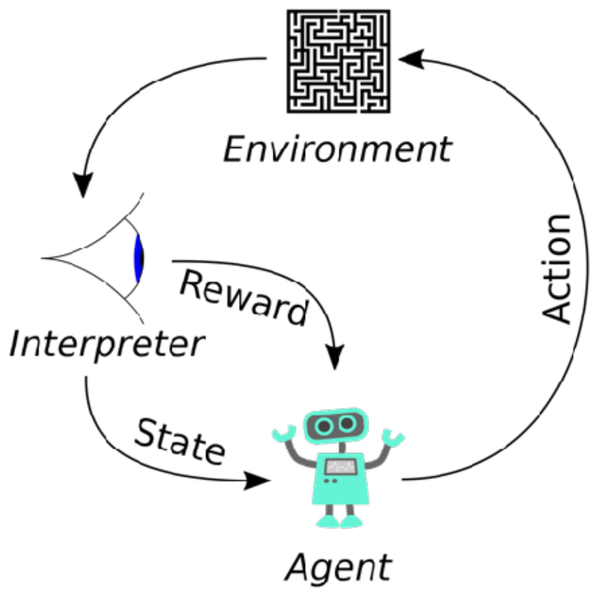
Sometimes need to sacrifice immediate reward to maximize total future reward, i.e., stop for red light even if goal is to get home in the shortest amount of time

Agent interacts with environment



- Sequence of steps t:
    - Agent executes action A_t
    - Receives observation O_t by interpreter/environment
    - Receives reward R_t
    - Environment receives A_t
    - Sends observation O_{t+1}
    - Sends reward R_t
    - Move to step $t + 1$ in sequence
- The record of (O_{i+1}, R_{i+1}, A_i) for all $i < t$ forms the **history** H_t
- The next step the agent takes needs to depend on the history - otherwise there is no learning
- The information available to the agent to decide the next action is given by the **state**

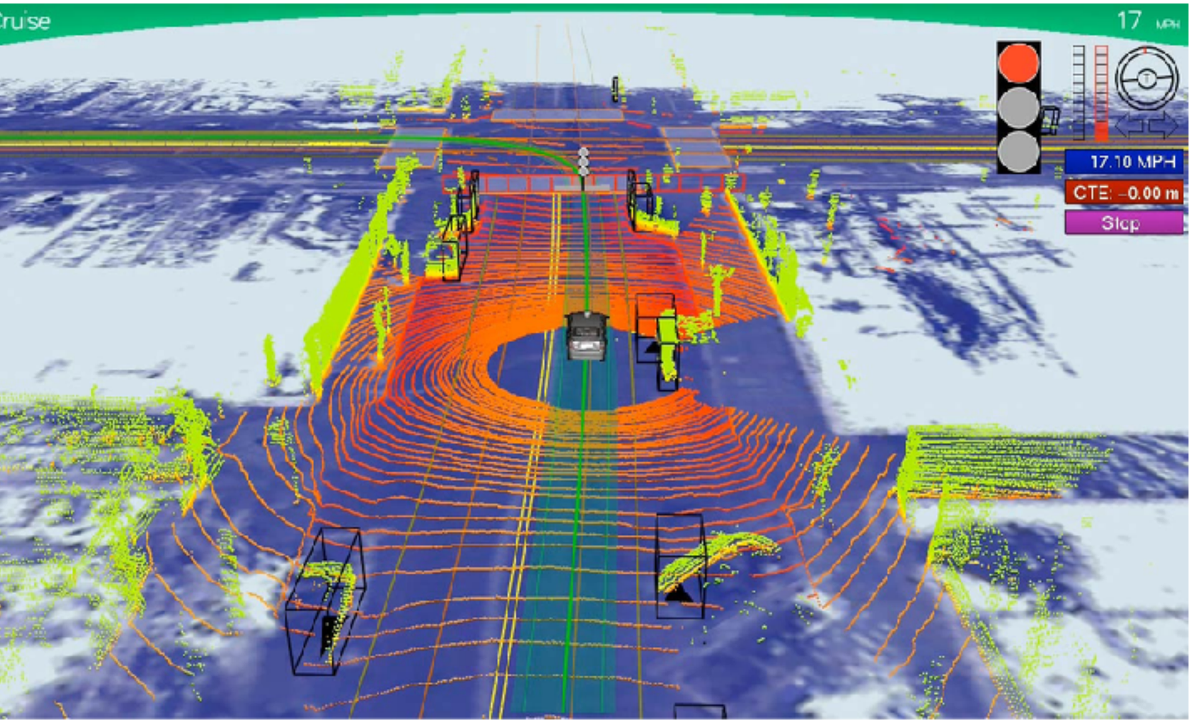
Agent interacts with environment



- Additional information may be in the **environment state** S_t^e, but this information is not available to the agent - the agent only receives observations O_t and reward R_t
- The specific information the agents acts upon is the **agent state** S_t^a, which is a function of the history $S_t^a = f(H_t)$

Markov state

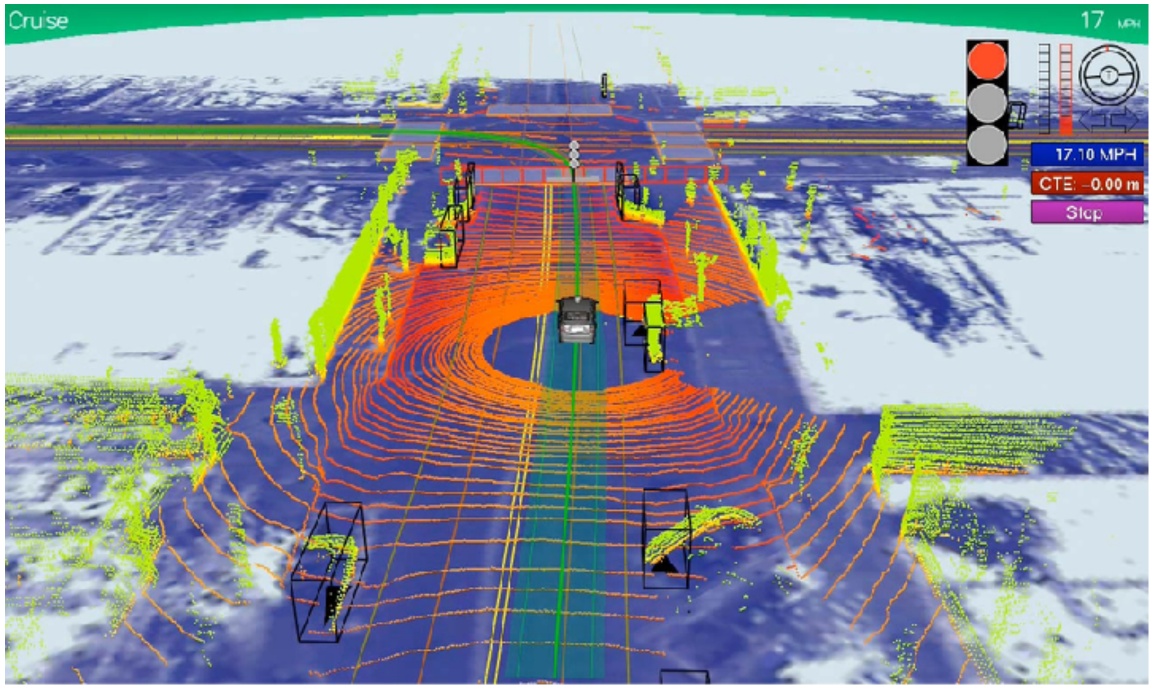
- A **Markov state** contains all **useful** information from the history
- Think of the Markov state as describing the present. The future will only depend on the present, not on the past steps that led to the present
- If a Markov state is known, the prior history is no longer relevant
- Here, the environment state S_t^e is a Markov state



e.g., whether the car should stop at red light does not depend on how long or from where the car has been driving

Observability

- **Full observability**: agent observes full environment state, $O(t) = S_t^a = S_t^e$. This is a *Markov Decision Process*
- **Partial observability** *(partially observable Markov decision process):*
  - Full environment state is not known to the agent
  - Agent must construct its own state S_t^a based on history H_t
  - This is the more interesting/relevant case

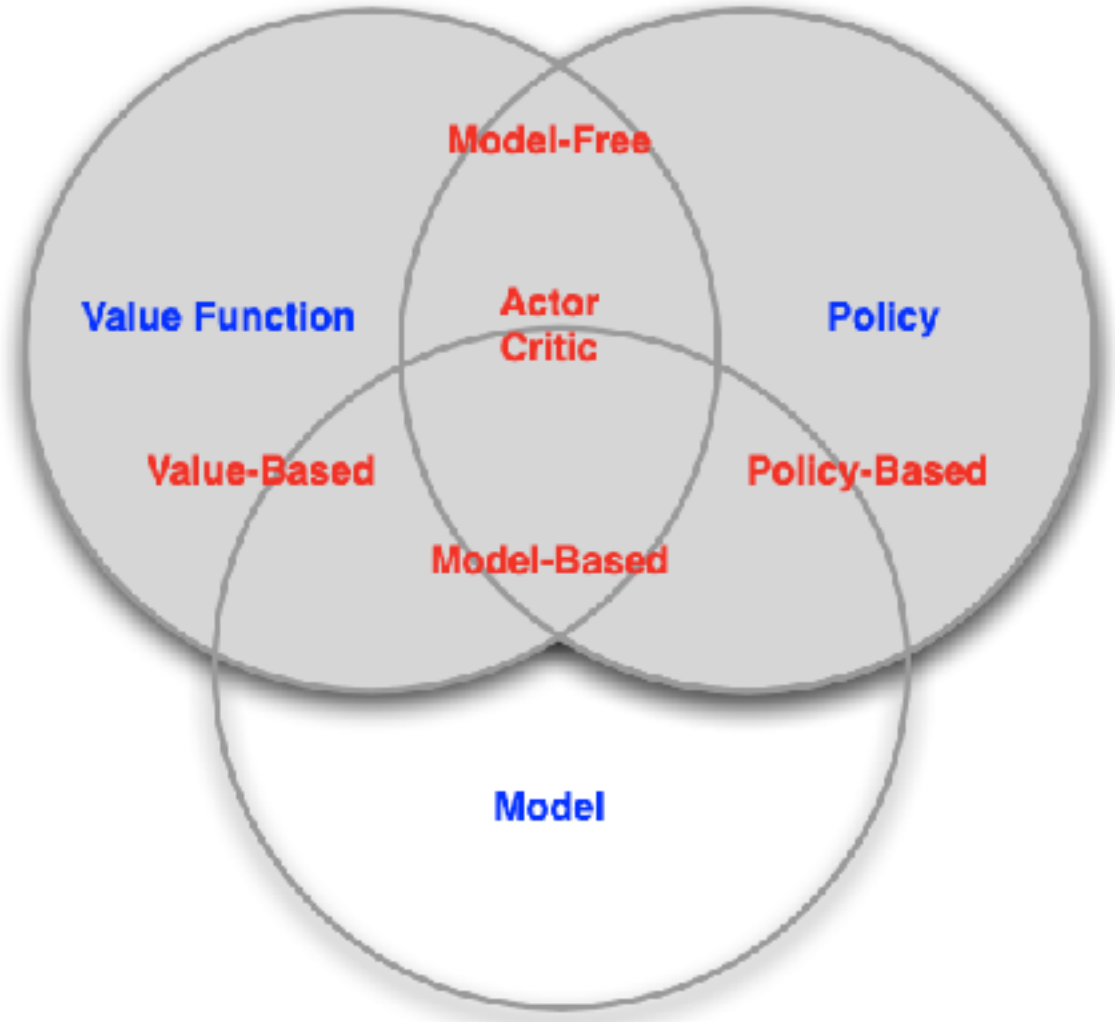


Car receives information from a limited number of sensors with limited range, angular vision

How to construct an RL agent?

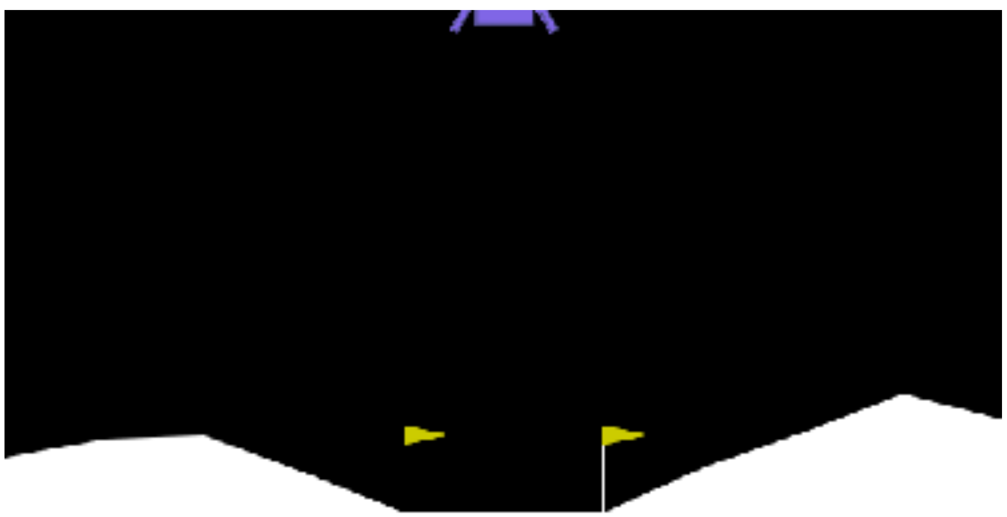
- RL agent needs some or all of these components
  - **Policy**: function that maps state to action. Policy can be deterministic, same state $\rightarrow$ same action or stochastic, with different actions with certain probabilities allowed for a given state
  - **Value function**: Prediction of future reward based on current state
  - **Model**: a representation of the environment; predicts response of the environment (observations and reward) and therefore the next state

Types of agents



from David Silver

Reinforcement learning sequence

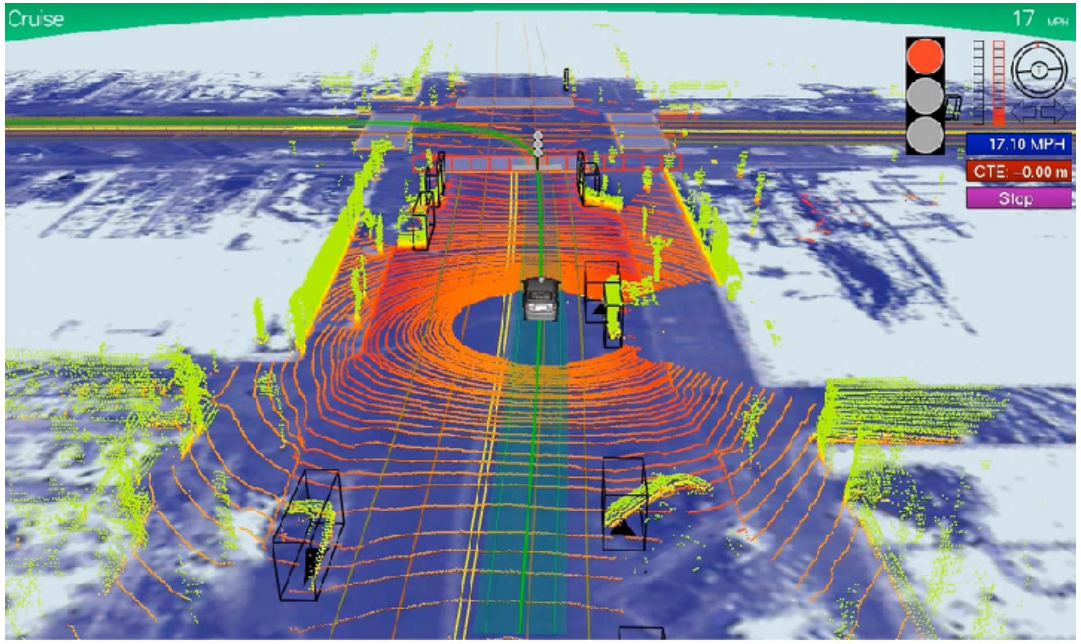


- Starting from unknown rules (unknown environment):
  - Agent executes action A_t
  - Receives observation O_t by interpreter/environment
  - Receives reward R_t
  - Environment receives A_t
  - Sends observation O_{t+1}
  - Sends reward R_t
  - Move to step $t+1$ in sequence

from David Silver

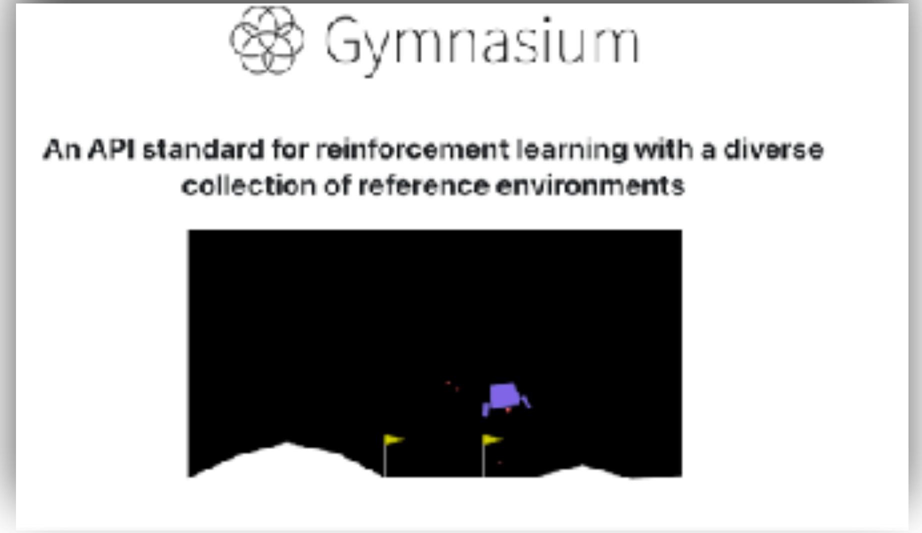
Reinforcement learning

- **Exploration and Exploitation:**
  - moves that uncover new information about environment (Exploration)
  - moves that based on current policy/value function are expected to maximize reward (Exploitation)



Take the same road home everyday
or
try a new route to see if it faster

RL example environments



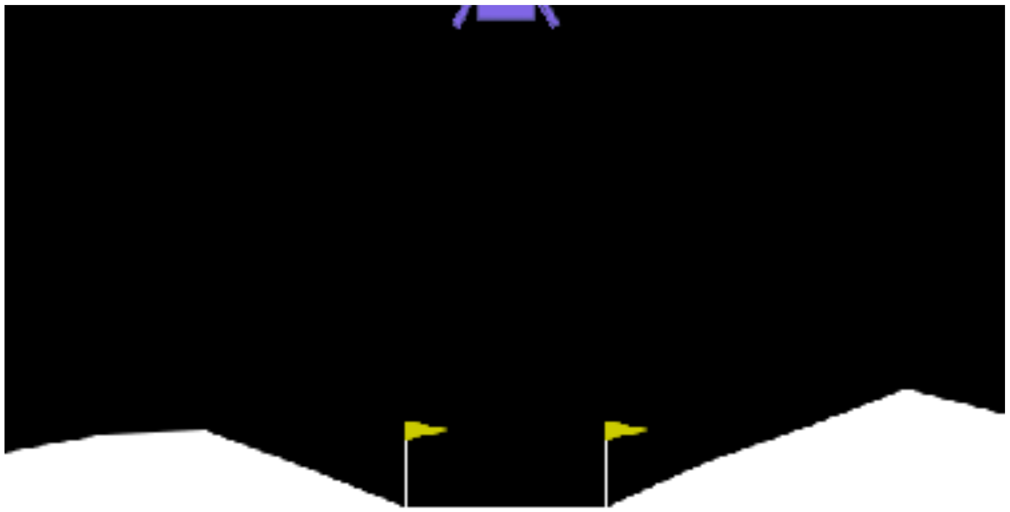
Gymnasium

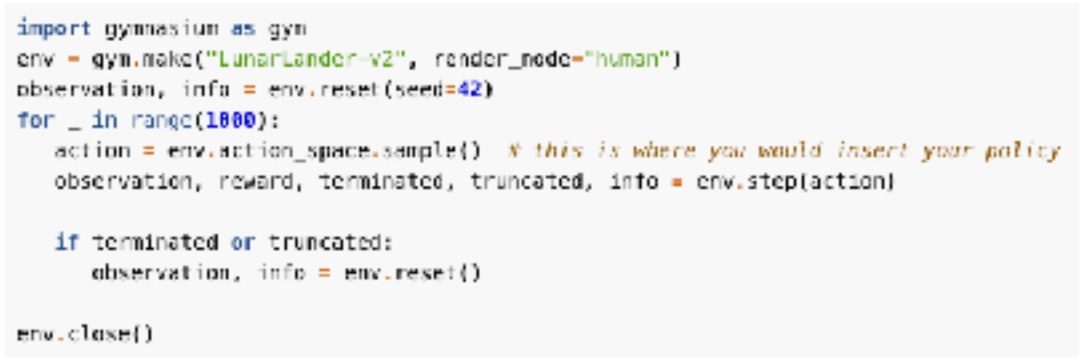
An API standard for reinforcement learning with a diverse collection of reference environments

https://gymnasium.farama.org/

- *Gymnasium* provides a number of virtual environments
- Represent different classes of reinforcement learning problems
- Agents can interact with environments through Python interface
- Will look at an example RL model interacting with Gymnasium environment

Example: Lunar lander





```
import gymnasium as gym
env = gym.make("LunarLander-v2", render_mode="human")
observation, info = env.reset(seed=42)
for _ in range(1000):
   action = env.action_space.sample()  # this is where you would insert your policy
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()

env.close()
```

To build an agent we need to know actions, observations and rewards…

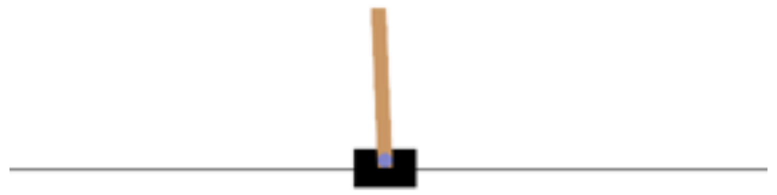
Example: Lunar lander

- Actions
  - 0: do nothing
  - 1: fire left thruster
  - 2: fire main engine
  - 3: fire right thruster
- Observation
  - 8D vector: coordinates, velocities in 2D, angle, angular velocity, two booleans for ground contact of legs

Example: Lunar lander

- Rewards
  - After every step a reward is granted. The total reward of an episode is the sum of the rewards for all the steps within that episode.
  - For each step, the reward
    - is increased/decreased the closer/further the lander is to the landing pad.
    - is increased/decreased the slower/faster the lander is moving.
    - is decreased the more the lander is tilted (angle not horizontal).
    - is increased by 10 points for each leg that is in contact with the ground.
    - is decreased by 0.03 points each frame a side engine is firing.
    - is decreased by 0.3 points each frame the main engine is firing.
  - The episode receive an additional reward of -100 or +100 points for crashing or landing safely respectively.
  - An episode is considered a solution if it scores at least 200 points.

Example: Cart Pole



- A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.
- Actions: Push cart to left or push car to right
- Observations: cart position, cart velocity, pole angle, pole velocity
- Rewards: +1 for every step t the pole doesn't tip over
- We will look at a PyTorch implementation for training a Cart Pole agent